Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of General Science

Per Holt-Seeland and Mathias Olsen

Automatic information security tests on a public cloud

Bachelor's project in Computer Engineering
Supervisor: Jonathan Jørgensen
May 2021

NTNU
Kunnskap for en bedre verden

Per Holt-Seeland and Mathias Olsen

Automatic information security tests on a public cloud

Bachelor's project in Computer Engineering
Supervisor: Jonathan Jørgensen
May 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of General Science

NTNU
Kunnskap for en bedre verden

Norges teknisk-naturvitenskapelige
universitet

TDAT3001 Bachelor

Automatic information security testing on a public cloud

Per Holt-Seeland
Mathias Olsen

May 20, 2021

Preface

The team was presented with an opportunity to participate in the creation of a task that the team would solve on behalf of Telia Company AS. The resulting task was created based on Telia's desire for the resulting product, and on the team's interest in working with information security. Additionally the team found it intriguing to work on cloud technology due to its significant relevance in modern software development. Information security and security testing is a vast field that this project will not come close to covering in the given time frame, leaving a large amount of potential future development. Due to this one of the project's goals will be to build a strong foundation for further development.

The team has enjoyed great freedom regarding choices and decisions, where the only decision that was predetermined was AWS as the cloud service provider of choice. From the very beginning Telia's representatives have set aside time in their schedule to conduct weekly status meetings, which has been great for consistent and frequent feedback on the team's work.

Despite of this project taking place during the COVID-19 pandemic, the project has not suffered any significant consequences. Digital meetings have to a large extent replaced the need for physical meetings and at this stage of the pandemic people are getting comfortable with digital communication platforms.

The team wishes to extend its gratitude to the Telia representatives Bjørn Vidar Smette-Øvergaard, Vaidotas Bakša and Kjetil Aune for providing a challenging and relevant task that has been very educational, as well as for taking time out of their days every week to conduct the status meetings. Furthermore the team wishes to extend its gratitude to Jonathan Jørgensen, our NTNU Supervisor, for his excellent feedback and swift communication.

Trondheim, May 2021
Per Holt-Seeland, Mathias Olsen

Task description

The goal of this project was to create automated security tests that could run on a public cloud. These automated tests are meant to reduce manual testing necessary and improve overall security of the applications. At the start of the project the goal for the security tests were to cover OWASP Top Ten.

After some development and foundation being made there were made changes to the goals of the project after deciding not all of OWASP Top Ten is automatable as a general solution. The changes were to cover some of the more easily automatable vulnerabilities from OWASP Top Ten, and then focus on authentication for the scans to be able to scan a large variety of applications.

The team were provided a cloud service from product owner, and further choices of tools and method is explained in the report.

Abstract

This report describes the results of the Automatic information security testing on a public cloud project. The product of this project is intended to improve overall application security on applications developed by Telia and to reduce the amount of time spent by Telia developers conducting security tests.

The product, config file and related scripts are hosted on AWS. In order to use the product, teams can fetch and modify config and script files before sending them to their AWS hosted container running an image of Kali Linux where zaproxy, python3 with a zap api package, and the necessary scripts to automate tests are preinstalled. Scans will be conducted on the scope specified in the config file and a report will be created in either HTML, JSON or XML. The report summarizes any potential vulnerabilities it has come over and can provide possible solutions to flagged vulnerabilities.

Configuring authentication in an automated manner proved to be a bigger challenge than expected, which in turn shifted priorities toward authentication and ended up limiting the amount of vulnerabilities the project was able to automate. Part of the reason why authentication became a bigger task than anticipated was the need to make the tests as general as possible for them to work on a wide selection of different applications.

The team started out with an idea to use SCRUM as the work methodology of choice, but soon came to the conclusion that Kanban would be better suited to the team size and task at hand. From Kanban the team utilized a Kanban board and roadmap to keep track of tasks and progress. Choosing a tool to handle the board and roadmap was done based on a combination of personal preferences on the team and the fact that Telia also uses Jira. The team is happy with their choice, as Jira has worked flawlessly and the user experience has been very good. Weekly meetings were held between Telia and the team, allowing for live adjustment of priorities as time constraints became increasingly apparent.

Reasonably frequent meetings were also held between the NTNU supervisor and the team, where the supervisor provided valuable pointers as to what the team should be focused on at different stages of the project.

This project has been highly educational for the team, due to the project exploring several topics neither team member had in-depth knowledge of at the beginning of the project. Both team members have expanded their knowledge on cloud services, security testing and authentication methods.

In retrospect the team is happy with the results of the project. However, the team could have had a more consistent workflow, reducing the need for long days and continuous weeks in the final stage of the project. Furthermore, the team should have put less effort into AWS research the first couple of weeks, perhaps until granted access to Telia AWS accounts, in order to prevent having to do the research twice.

Contents

1 Introduction and relevance

Information security is a large and important field that constantly evolves. Hackers, both ethical and malicious, are constantly looking for vulnerabilities that can be attacked or exploited. While ethical hackers disclose the vulnerabilities for developers to fix them, hackers with malicious intent may cause severe harm to a business, institution or even a private person.

Leaked personal information or sensitive company data is a significant threat in a digitalized society and every company that offers digital services or products carry a responsibility to do what they can to make sure their services or products are safe. A significant part of the effort to make sure services and products are safe, at least from known vulnerabilities, is conducting security tests. This is a time consuming task that requires focus from developers if they are to be conducted manually.

This is where automated security testing comes in, not only could successfully automated security tests lead to increased accuracy in security testing but also save a significant amount of time for developers, allowing them to focus on development rather than repetitive and time consuming security test. Focusing on these benefits, this project will try to explore *how security tests can be automated in order to reduce manual testing.*

1.1 Automating security tests

This project will aim to create a general solution that will work across several applications rather than one specific application, automating security scans that are reasonably automatable within the project time scope. Certain security vulnerabilities are too unpredictable to have tests automated within the timeframe of this project or as a general solution.

1.2 Report structure

Chapter 2: Theory: The theory chapter will cover necessary theory needed for this project. These include theory about security testing, common vulnerabilities, and some theory about tools used in this project.

Chapter 3: Choice of technology and method: This chapter covers choices for technology and methods made along the way, and why these were chosen.

Chapter 4: Results: The results chapter covers the result over the project and how these compare to the requirements and goals set for the project.

Chapter 5: Discussion: This chapter will discuss the project as a whole. Decisions made along the way will be discussed whether it was a good choice, or possible alternatives that should have been done instead. Limitation and positive elements will also be discussed, and a conclusion for the project along with possible future work that can be done.

1.3 Terminology and abbreviations

Fuzzing - Repetitive action trying different combination of data for input to a program

XSS - Cross-Site Scripting, security vulnerability

ECR - Elastic Container Repository owned by AWS

ECS - Elastic Container Service owned by AWS

SSO - Single Sign-On, authentication method

CSRF - Cross-Site Request Forgery, security vulnerability

JWT - Json Web Token, used for authentication and authorization

CI/CD - Continuous integration/continuous development

Spider - Tool used for passive scanning and finding connected URLs

S3 - Cloud Storage owned by AWS

ZAP - Zed Attack Proxy, security penetration testing tool owned by OWASP

AWS - Cloud service Amazon Web Services

2 Theory

2.1 Testing

Testing in development is meant to help discover bugs early in the development and will also notice if bugs appear during maintenance later. There are many benefits to discovering bugs early in the development. The cost of fixing the bug will increase the longer it takes to discover it, this is because the bug might affect other parts of the application, which also needs to be changed and it will take more time to discover the cause of the bug later when the developer does not have the code fresh in mind. Application security and product quality will also benefit from testing, where the sooner a security vulnerability is discovered the less time there is for attacker to take advantage of the vulnerability. Overall product quality will also improve if consistent tests discover bugs before they reach production.

2.1.1 Security testing

Application security is becoming increasingly relevant and there are different ways to test security. Before testing the security it is important to know which vulnerabilities are common, how these are exploited and how to defend against them. Security vulnerabilities are categorized into two categories; technical and logical vulnerabilities. Technical vulnerabilities is when an attack can exploit a technical mistake, for example lack of input sanitation. These vulnerabilities are easy to automate with penetration testing tools, reducing the need for manual testing.

The logical vulnerabilities are logical mistakes in the application which can be exploited. A common logical vulnerability is mistakes in access controls, allowing users with limited rights to access files or actions not permitted to them. This is a logical vulnerability because the application fails to validate access rights before granting access. While the logical vulnerabilities are very different they are commonly more difficult to create automated tests for and require more manual testing from seasoned developers experienced with the application. [13]

Automated tests are good at finding technical vulnerabilities which means setting up a good automated testing system will not only save time for developers but will also increase accuracy of the tests if done right. However this does not remove the need for manual security tests. Not all vulnerabilities are possible to automate and double checking the automated security results periodically can uncover mistakes in the automated tests if there are any.

2.1.2 Unit tests

Unit tests are testing individual components to make sure every method works on their own. By testing all components individually the tests save time for the developers to specifying exactly which methods is not working as it should. However these tests will not discover logical mistakes from how components work together, as components are only tested individually. [14]

Unit tests are common to implement and is recommended to cover a larger percent of the application. If the application relies on CI/CD unit tests becomes critical to discover mistakes so the CI/CD can stop the code change before pushing.

2.1.3 Continuous Integration/Continuous Deployment (CI/CD)

Continuous Integration focuses on integrating development branches frequently to avoid merging larger branches and get integration issues. Developers create tests to make sure the main branch works as it should and CI will run all the tests on each merge to main. CI will then discover possible issues with the merge and stop the merge if the tests does not pass.

Continuous Deployment takes CI to the next step and will automatically deploy the new changes to the customers. This is one of the most effective ways to make changes on feedback

from customers but will also make errors more critical since everything is happening automatically and no developers are part of the deployment process to discover these errors. The only thing stopping the deployment is if the tests fail, thus it is important to have sufficient test coverage to uncover all possible errors[1].

CI/CD can also be integrated into automated security tests by stopping a merge if the changes contain security errors. However if the application with CI does not get new changes these security tests will not run. If a large amount of time goes without running security tests, new updates to scanning tools and tests could have discovered new vulnerabilities, but the tests does not run to find them.

2.2 Security vulnerabilities

2.2.1 OWASP Top Ten

OWASP (Open Web Application Security Project) is an online community which works to spread awareness around web application security. They do this by producing and sharing articles, research, tools and other technologies for testing and improving web security. One of the more well known documents produced by OWASP is the OWASP Top Ten list. This list aims to spread awareness about the most critical security risks which a lot of applications is not secured from.[23]

This list is updated every few years to make sure the list is up to date on new security risks. There are also some security risks which have stayed on the top ten list for a longer period of time. Some of these are Cross-Site Scripting and SQL Injection. The vulnerabilities present on the OWASP Top Ten have their spot because of how common the vulnerability is, and how severe the consequences of a successful attack is.

2.2.2 Cross-Site Scripting (XSS)

XSS is an injection type attack where the attacker uses the websites input fields to send malicious code, usually in the form of scripts. These scripts will then attack users trying to use this website to steal their personal information or make them do an unintended action.

The earliest names given to the different types of XSS attacks were Reflected and Stored XSS[26]. Reflected XSS are an attack usually being passed through a link or query where the victim will receive a link to the website containing the script. When the victim now clicks on the link the script will gain access to steal their personal information through cookies or other stored variables.
Stored XSS attacks are more persistent attacks where the scripts instead are injected into the server and does not require the user to click on a specific URL. The script which is saved on the server will be a permanent feature until the developer removes it. Whenever a user accesses that website while the script is active, the user's information like cookies will be stolen by the script[11].

In 2005 a new type of XSS was defined by Amit Klein[12]. This type is called DOM Based XSS because the goal of the attack is to change the DOM environment in the victims browser. This causes the client side code to run in unexpected ways, allowing the attacker to obtain the information he or she wants. The main difference from the other types of attacks is that the DOM based XSS does not change the HTTP response received from the server but rather change how the client handles the received response.

These types of attacks can also be defined into server and client side XSS, since it is proven that reflected and stored XSS can be used both on server and client side. The DOM Based XSS is considered a subset of the client side XSS.

2.2.3 SQL Injection

SQL Injection is also an injection type attack, but instead of injecting scripts, the attacker is injecting SQL queries to get information or alter the database. This type of attack has been at the top of the top ten list for a long period of time, both because of how common the vulnerability is, but also because of the severe consequences of a successful attack. [25]

A successful SQL injection can do a lot of harm based on the intent of the attacker. Stealing sensitive information from the database is normal way to use the attack and can go undetected a long time if the website has insufficient logging. This will then allow the attacker to continuously steal information from the database until detected. Through the SQL Injection the attacker can potentially steal login information from an administrator and may be able to alter the website through the administrator's user. Another critical use of the SQL Injection is to alter or delete large portions of the database, which can cripple a running website for a longer period of time. A lot of important information stored in the database can also be lost for good unless the database has a backup.

2.2.4 Brute force attack

Brute force attacking is a way to break username and password combinations by attempting to log in to a preferably known user's account using a large amount of different passwords. The passwords tend to be fetched from a list of commonly used passwords or common words in various combinations of lower and upper case letters and numbers.

2.2.5 Broken Access Control

Broken Access Control comes from mistakes in enforcing rights to functionalities and data. An attacker can exploit this vulnerability by performing functions and accessing data the attacker have no rights to use. Depending on how broken the access controls are the attacker can exploit larger part of the application, and basically become an admin to the site. [17]

This one of the most common logical vulnerabilities and have been present on the OWASP top ten list for a considerable amount of time. Since it is a logical vulnerability it is a lot harder to automate security testing for it, which makes it harder to discover and remove. There are automated security scans that can discover if access control is completely absent from the application, but those scans can not discover whether existing access control works as intended or not.

If an attacker manages to exploit this vulnerability it can be hard to reverse the damages since it is hard to know exactly what the attacker has done. After the vulnerability is fixed, there should be an evaluation of rights given to users, in case the attacker have changed the access rights of other users.

2.2.6 Insufficient Logging and Monitoring

Logging and monitoring of attempted security breaches is important to stop similar attacks in the future, or notice successful breaches. If the logging and monitoring is insufficient an attacker can plan and execute his attacks over a larger period of time, as his attacks go unnoticed. These attacks can for example be penetration testing tools running scans on websites looking for vulnerabilities. With enough time of undetected security scans, new vulnerabilities may be discovered and exploited. [21]

A good way to test for insufficient logging and monitoring is to set up automated security tests, and check if these attacks are logged. Even if they are logged, an active attack like penetration testing should be considered for triggering alerts to developers, in order to make sure actions are carried out promptly.

2.3 Web applications for security testing

Some web applications are intentionally designed to have several security flaws. These are made to give developers an application to test for known security errors and see if they can find them. This is good practise before testing on their own websites, to make sure the test will actually uncover security flaws.

Some of the examples for these applications are Damn Vulnerable Web Application (DVWA), Juice-Shop and Bodgeit. These applications are popular in the security testing community and contains several well known security flaws. They also provide a way to change security level so you can test the application whether you're experienced hacker or new to the game. These three applications also use different authentication methods. Due to this, scanners need support several authentication methods to be able to scan all.

2.4 Authentication

There are several ways to authenticate a user before giving access to an application. Since the purpose of authenticating a user is to prevent users with malicious intent from gaining access rights they should not have, the authentication should be secure from being used in malicious ways as well. This is why there have been developed different types of authentication, each with the purpose of defending against malicious attacks.

2.4.1 Basic Access Authentication

Basic access authentication is regarded as one of the simplest and least secure authentication methods. When a user is getting authenticated through this method the user provides the username and password, and these credentials are added as a HTTP header then sent as a login request. What makes this method vulnerable to security breaches is that the message itself is only protected by base64 encoding, which is not hard to decode. Which makes the method vulnerable to man in the middle attacks, where the login message is caught by a malicious user. [30]

This problem can be solved by using SSL to secure the packet, but there are other security vulnerabilities where the encrypted password is stored temporary or permanently in the web browser, which then could be reused by malicious users by performing for example a CSRF attack.

2.4.2 Cross-Site Request Forgery tokens (CSRF Tokens)

CSRF attacks is when an attacker induces an authenticated user to perform an unintended action. This action can vary based on the different rights the induced user possesses, but a normal action is to make the user change his email address, enabling the hacker to take control of the account. For a CSRF attack to be successful the attacker needs to perform a predictable request. A predictable request is a request only containing information which is retrieved from the induced user's cookie, or information the attacker can find on his own. [19] [27]

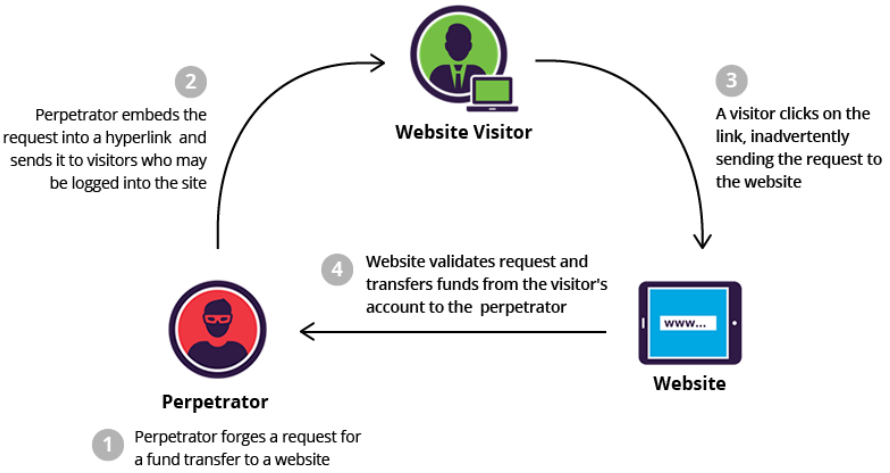


Figure 2.1: Example of a CSRF attack [10]

If the application is using CSRF tokens every request will be unpredictable because a unique token that is not present in the cookie will be required. Since the request requires this token to be valid the attacker can not recreate the request, making it impossible to perform a CSRF attack.

The CSRF tokens should be generated to be highly unpredictable, in order to prevent an attacker from being able to predict a correct token through brute forcing many different combinations. For transmitting the token to the user it is normal to have a hidden field in the HTML, which is then retrieved and submitted through the POST message. [28]

2.4.3 JSON Web Tokens (JWT)

JWT is a compact security token used to authorize an user. After an user has provided credentials and successfully logged in, a token will be returned. This token will then be used to authorize the user in future requests. [3]

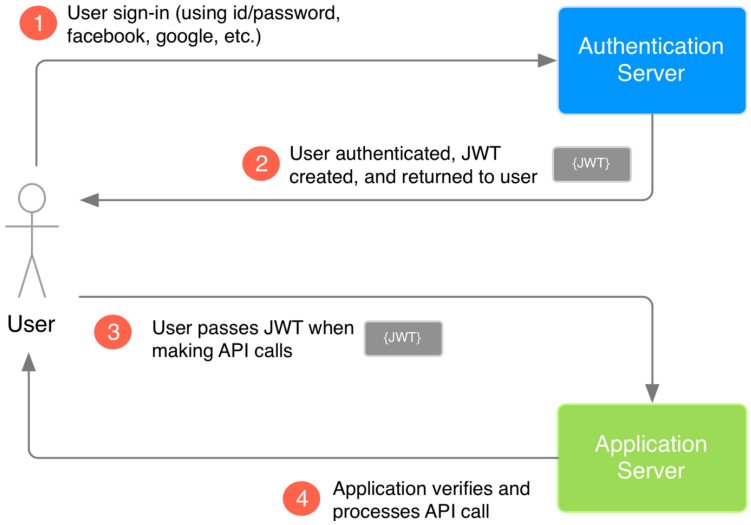


Figure 2.2: How JWT is used [9]

When information is sent with a JWT, the information can be trusted because the JWT is

signed. The signature on a token is made by encrypting the header and body of the message, and when decrypted the signature should match the content of the message. If the message is intercepted and changed, the signature wont match the content, and it is safe to say the message has been changed. Even though the message can not be changed without the recipient knowing, the content can still be read. This means it is still important to ensure the message is not intercepted. [2]

2.4.4 Single Sign-On (SSO)

With SSO a user can log in to several applications through one authentication process. SSO has become more popular over the last years where companies like Google provides one authentication progress for all their applications, but also provides SSO through Google for other applications.

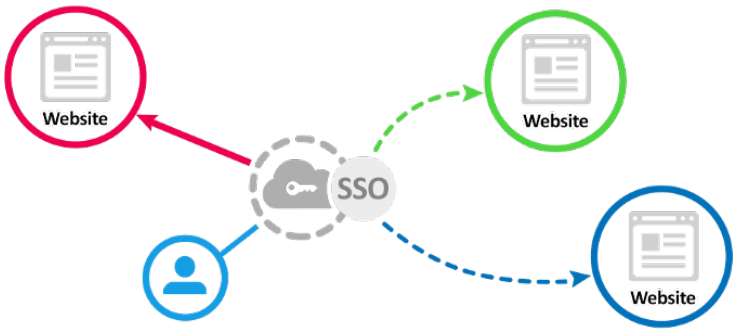


Figure 2.3: SSO connecting to several applications [29]

SSO works through use of sessions, with different types of sessions that serve different purposes. A local session is being maintained by each individual application, and will not interfere with other applications. If SSO is enabled a larger server side session is used over multiple applications. This session is what makes SSO possible, by checking if a user is already logged in to another application through SSO, removing the need to log in again. Sessions are also used for handling different identity providers like for example Google or Facebook. [4]

2.5 Zed Attack Proxy (ZAP)

Zed Attack Proxy is a security penetration tool developed and maintained by OWASP. It is a free, open-source tool made to help help users to find security flaws. ZAP can be used through its graphical user interface but it also has an API which can be used instead. [20]. For creating automated security tests the API is needed, but the UI is practical for manual penetration tests.

2.5.1 Context

A ZAP context is way to configure scans made with the ZAP tool. The context is used to define the scope of scans by using regex to include and exclude URLs. Scans will stay inside the scope defined by the context and not actively attack websites outside the scope. Authentication is also defined by the context, where you will provide information regarding what authentication method and login credentials to use. [18]

There are several ways to create a context in ZAP. For first time users it is recommended to create the context manually in the UI to explore the different options given. All contexts can also be imported and exported, and the API also provides a way to create new or alter existing contexts.

Contexts also contains users specific to different applications. The users are the ones to contain the login credentials, and it is possible to have different users, with different rights in the same context. This may be necessary, for example in order to test if a user without admin rights can access privileged data and actions.

2.5.2 Passive scan - Spider/Ajax spider

A passive scan is a non-aggressive way to scan a website. The scan is usually done via a spider or crawler and can be used to map a website and find linked URLs. Since these scans are defined as non-aggressive they can be used on a website without the fear of performing unintended attacks. This means using a spider to scout a website before performing an active scan is a good way to prevent attacks outside the desired scope. If the passive scan finds a website you do not want to attack, the scope needs to be adjusted before performing the active scan. [24]

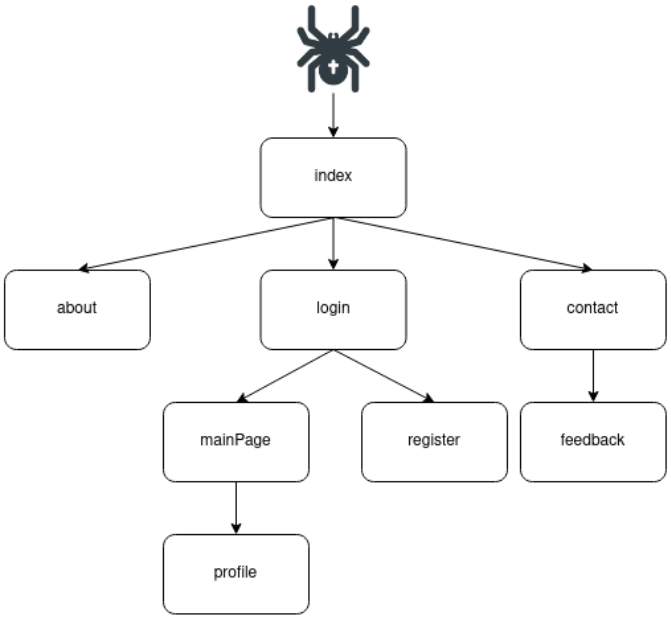


Figure 2.4: Example path of a spider

ZAP provides two different spiders to perform passive scans. The two choices are regular spider and AJAX spider. The main difference between them is that AJAX is is able to crawl applications written in AJAX in far more depth, and is recommended to use on more modern applications made in Vue.js, Angular.js and React.js. [22]

2.5.3 Active scan

An active scan is an aggressive way to scan a website. This means the scan will actively attack the website to look for security flaws. There are several different security flaws an active scan can look for, including SQL Injection and XSS mentioned above. Since the active scan is performing an actual attack on the website it is important to be careful where to scan. Defining a scope of the attack and running a passive scan first are good measures to prevent attacks on unexpected websites. [15]

2.5.4 Authenticating scanners

Authentication is an important part of running security scans, because the scanner need access to the whole application in order to scan everything. The scanners are meant to scan

several different applications, possibly with different authentication methods. This is why configuration of the different authentication methods being used is important. [16]

ZAP provides different out of the box methods that can be used to authenticate a scanner as well as support for authentication scripts. This means that even if an application uses authentication not supported out of the box, scripts can still be created to authenticate the application.

ZAP calls this the script based authentication. Where the user defines which variables are needed, what POST message needs to be sent, and when the POST needs to be sent. The scripts also have other uses than authenticating the less used methods. When CSRF tokens are being used in authentication, ZAP needs a way to find this token and add it to the POST message. This can be done with a script.

Form-based authentication is another method provided by ZAP. This method of authentication is similar to Basic Authentication where both methods lacks in security as the credentials are submitted in plain text. One of the differences however is that form-based authentication requires the user to actually fill out a form, while with basic authentication it is possible for the browser to retrieve the information through other means.

ZAP also supports the use of JWT authentication, this can be done through different methods. JSON-based authentication is a feature where all the credentials are sent as a JSON object. Combining this with a script to receive and save the returned JWT and then attach the JWT to future requests, will authorize the scanners to access content that requires authorization. The script made to retrieve the JWT can be written as HTTP sender script or a session management script.

2.6 Cloud Computing Service

Cloud computing [8] is technology that offers on-demand IT resources online. The use of cloud computing services comes with many benefits. They tend to be billed by usage, which means there is no need to pay for more resources than is actually consumed. They scale depending on the users demands and can even be set up to scale automatically. By provisioning cloud services the user does not need to acquire, manage or maintain hardware needed to run computing services. Users can also opt out of maintaining servers completely by utilizing cloud computing services that are maintained by the cloud provider itself. For companies that serve a global network of customers, provisioning services from a cloud provider with a global network of data centres, enabling hosting of mirrors around the globe in order to reduce latency for geographically distant customers.

A cloud computing service will by its nature be remotely hosted, in many cases even on foreign soil from the customer. As a consequence of this, hosting critical functionality or data on-premises may be required in some cases for security reasons.

2.7 Amazon Web Services (AWS)

AWS [5] is a cloud computing platform made by Amazon. AWS offers a wide range of cloud based services, allowing customers to select specifically tailored solutions suitable for their needs. At the time of writing AWS held approximately 32% market share, making AWS the most used cloud service provider ahead of Azure at approximately 20% and Google Cloud at approximately 7%.

2.7.1 Amazon Elastic Container Service (ECS)

Amazon ECS is a container management service that is used to manage, run and stop containers. ECS can be used with Amazon EC2 or AWS Fargate, and can manage the entire container lifestyle from start to stop.

2.7.2 Amazon Elastic Container Repository (ECR)

Amazon ECR is a container repository where images can be stored for use with Amazon Elastic Kubernetes Service and Amazon ECS. Using ECR eliminates the need for operating a container repository, and being part of the AWS ecosystem it of course integrates very well with other AWS services. Updating an image stored in ECR is easily done by pushing the updated image to ECR. With the containers spinning up from images in the ECR this also means containers are always spun up using the updated image.

2.7.3 Amazon Fargate

AWS Fargate[6] is a serverless compute engine for containers. As opposed to EC2, which is a cloud virtual machine service where all management is handled by the user, Fargate hosts an application or service without the user having to manage the underlying infrastructure. Fargate can automatically scale to suit the resource needs of the instance(s) of an application or service. When using Fargate, only actual runtime of containers or services are billed. In most cases there will be a minimum of containers running, to avoid high latency which could lead to bad user experiences or even request timeout. However, in special cases where latency is not an issue, Fargate can be set up to not have containers running standby, but spin up on request and shut down when completed, potentially reducing runtime cost.

2.7.4 AWS Lambda

AWS Lambda[7] is a serverless compute service. This allows customers to run code without provisioning or managing servers or containers, a function will simply run when it is called. This has advantages when it comes to maintenance as well as cost. This service is well suited for relatively simple functions that have short run times.

2.8 Docker

Docker is a platform that allows running of applications in an isolated environment referred to as a container. By using a Dockerfile docker allows automatic building of an image, from which a container can be spun up, where docker pulls the latest image from a repository, for instance dockerhub and builds the image as specified by the Dockerfile. This allows specifying that the image should update packages, import any required packages on build and copy files from the host system to the image, resulting in a ready-to-use image that is fully updated and has the specified packages installed.

3 Choice of technologies and methods

3.1 Application

3.1.1 Authentication

Decisions about how the scanners are authenticated is a very central part of the project. Which methods of authentication that get supported decides which applications these automated tests are viable for. Getting as many authentication methods included in the project would create a good foundation for running these tests against several application, allowing future work to focus on creating new tests.

Another factor to consider is the ZAP tool and which authentication methods are easily supported by ZAP. ZAP provided some guides on their official websites [31] on how to authenticate specific applications through some of the methods mentioned earlier in Authentication 2.4. Since the methods covered in the guides were well used, these were the starting point of authentication coverage.

Decisions were made to cover CSRF tokens and JWT because of their popularity, the level of security they provide, and after wishes from product owner form based authentication was also added as a simpler method. However, all applications are using authentication methods in their own way, so configurability of the authentication was important. The solution to this was using a larger configuration file for all variables regarding to authentication and context, and providing different templates for how the variables should be set against different methods.

```
#Authentication methods. Set True or False
UseBasicAuth = False
UseCSRF = True
UseJWT = False
```

Figure 3.1: How to specify authentication method

After the authentication method is specified like shown in 3.1 the program will define the authentication method in context, and decide how the login data is handled. Providing the login URL and POST data for the login is also crucial for logging in, and is also provided in the configuration file.

```
#Post data for login
PostData = username={%%username%%}&password={%%password%%}
#URL for login
LoginUrl = http://localhost:8082/bodgeit/login.jsp
```

Figure 3.2: How to set login URL and POST data

Since scripts also played an important role in authentication and they also vary depending on the needs of the application they also needed to be configurable. To give this option the team provided default scripts which would run with certain authentication methods, but remains well documented, and is susceptible to change.

3.1.2 Scanning

The security tests are done by penetration scanning. By changing the configuration file the user can define a starting URL for the scans, and provide URL regex like shown in figure 3.3 to define the scope where the scans will attack. The security tests will first run a passive scan

to map out the found URLs within the scope so the user can verify that the scans will not attack something he does not want to attack. After the user has verified this, an active scan will start running active penetration tests attempting to find vulnerabilities.

```
#URLs to be included in the scan
IncludeInContext = [
        "http://localhost:8082/bodgeit.*"
        ]

#URLs to be excluded from the scan
ExcludeFromContext = [
        "http://localhost:8082/bodgeit/logout.jsp"
        ]
```

Figure 3.3: Specified URL regex from Bodgeit template

As this project is focusing on a general solution viable for testing several applications the focus was set on technical vulnerabilities. These vulnerabilities were mainly SQL injection and Cross-Site scripting, but these are included in a larger scan for injection type attacks. These were chosen because of their high placement on the OWASP Top Ten list, and because they are well known vulnerabilities to cover with automated security tests.

3.1.3 Scan results

Security alerts found by the security scans are continuously added to a report which can be presented as HTML, JSON or XML. This report contains a summary of all the different vulnerabilities found, where they were found and number of instances found. The alerts are put into four different categories: high, medium, low and informational. The team decided to use ZAP's existing alerts because they are explained well, and provide possible solutions to remove the vulnerability.

Another alternative to ZAP's report could be to create a new report based on the report given from ZAP, but remove the parts deemed unnecessary. This would make the report more specific, but could potentially remove important alerts. Using ZAP's report was decided to be the safest route, and the developers could decide for themselves which alerts are important to their application.

After the scan is complete, an alert will be triggered if any high security alerts were found during the scan. This alert is currently only made in the running console, so it wont do much when the tests run automatically, but can be integrated into for example Slack for better notifications.

3.1.4 Scanning intervals

There are two options for possible testing intervals that were considered. The first option is integrating the automated security tests with CI/CD, which will run the security tests whenever a change is pushed to the main branch. This is a viable option as long as changes are made to the main branch consistently. Security tests and penetration testing tools will be updated periodically, so it is important the tests continue to run even if there is not any changes to the application. This means that if the tests run on CI/CD, and there are not any changes for a considerable amount of time, the tests still have to be ran after changes made to tool or tests. This can be solved by forcing an automated security test to all applications using the tests, whenever the test or tool are updated.

Another option considered is running the tests on specific intervals regardless of changes made to the application. This solution will remove the need for forcing new tests after updates because the tests will run periodically anyway, however there may be gaps in time where new changes got security vulnerabilities, but the tests does not find them right away. More resources will also be spent on applications without changes, and without changes to the tests. Which should provide the same result every time.

Both are good options that would work, but for this project a CI/CD solution were chosen. This was because of further development can make CI/CD far better with integrating terraformed infrastructure which is used by project owner. This terraformed infrastructure would provide a way to force a security scan to run across all relevant application, and therefore remove the disadvantages of the CI/CD solution.

3.2 Configurations

Since these tests are meant to be implemented into multiple teams with different applications, the configuration of the tests are important. These applications may also use very different methods of authentication so supporting options for different authentication was a significant part of the project, and everything should be done through one configuration file. The context for the scan is created based on this file, and will update itself if there are any changes to the file. One of goal for the automated security scans is to make it easier for teams to test security, without the need to know too much about it. Removing the need to make changes to scripts to make the test work for specific application will make it easier to use and implement.

There are also variables not included in the configuration file, but can be changed if the user decides it is necessary. A good example of this is the attack mode of the ZAP application, which is set to protective, and can only be changed in the script. The different ZAP modes decides how aggressive or how cautious the attack are performed, and the protective mode is very cautious and will prevent any and all attacks on URLs outside specific context regex.

3.3 Script organizing

The script are organized around a main script which starts the application. This main script will retrieve all variables from the configuration file, and then look for an existing context matching the variables. All context related methods are located in a createContext script and will be called for creating context and user if needed.

After the context and user is created with content from the configuration file the application will run scans. All methods relating to the security scans are located in a runScan script. These scripts take the context, user and target as parameters to perform the security scans. Currently the script contains two passive scans using different spiders, and one active scan. With this script setup it is easy to create security scans in existing scripts, or create new scripts to test additional security vulnerabilities. Only changes necessary to the main script will be to add additional config variables, and run the new methods made in other scripts.

3.4 Technologies

3.4.1 Version Control

For this project the decentralized service Git is the version control of choice. This decision is based on Git's wide array of useful features. Git provides a practical way of sharing code between the developers of the project, including functionality to perform comparison where there are conflicts and to some extent automatically resolving conflicts. Git also provides functionality for reversal if a change or an update introduces bugs or breaks the system. Reversal can be done by rolling back to a commit that is known to work. Another possible way is to go through the changes made by the commit that introduced bugs line by line, as git

presents the changes of every commit in a very readable way. GitHub has been used to host the repository of this project.

3.4.2 Programming language

The programming languages that were considered for this project were Java, Python and Javascript. All these languages provide good libraries for security testing and implementation of different tools which might have been used in this project, or may be included later.

Python ended up being the choice of language due to its scripting properties which easily could be implemented with the main tool used for this project, ZAP. Python had a simplified ZAP API which made the code easier to read, and makes the connection to the ZAP API simpler. Python's popularity and easy-to-learn nature may also make further development a less complex task. Python libraries are frequently upgraded and libraries receive frequent additions, indicating Python is likely to remain relevant within the field of cyber security.

3.4.3 Virtual Machine (VM)

When security testing is conducted it is preferable to contain the application within a virtual machine, in order to limit potential damage if something goes wrong or the testing causes damage to the application it is attacking.

During development of this project Kali Linux has been the virtual machine image of choice.

Kali was chosen because the VM is tailor made for penetration testing, ethical hacking and other cyber security tasks. Due to this the VM comes with a wide selection of testing tools pre-installed, including the ZAP tool utilized in this project. All this makes Kali the perfect choice of VM for this project.

3.4.4 Penetration testing tools

The choice of penetration testing tool used in the project is an important decision and stood between Burp Suite and ZAP. ZAP is the open source penetration testing tool made by OWASP and is free to use. Burp Suite is made by PortSwigger and gives the choice between a free and premium edition of the program. Both programs provide ways to scan applications for security vulnerabilities and are popular in the software security community.

The decision was made to use ZAP because it is a free open source program, and the product owner preferred open source over commercial subscriptions. ZAP works well with the choice of language and has an easy to use ZAP API alongside the ZAP UI. The downsides of ZAP versus Burp suite is that burp suite provides more features than ZAP, but requires a premium subscription to unlock these features. Another advantage of an open sourced application is that it often has faster development, and since it is dependent on volunteers and feedback the users have more to say in the application's development.

3.4.5 Maintaining the penetration testing tools

In order to maintain the security and reliability of the scanning software, it will need to be kept up to date. There are several possible approaches to this with their own benefits and disadvantages.

A reactive approach has the benefit that it does not require user input, which in turn has the benefit that there will be little to no delay in updates as they are released. A disadvantage to this approach is that it may cause issues. If an update is released with a bug that is unknown at the time of release or the release breaks the scanning tool or its setup, security may be compromised. Furthermore it may cause issues if an update is run at a time the system experiences heavy traffic, disrupting running tests. The disruption issue can be mitigated by running updates as a cronjob outside of working hours or at times of low traffic. The bug issue

may be mitigated by pinning the application to a version number, in which case only minor patches within the same release will be applied, often to solve bugs or minor issues.

A detective approach may be beneficial with regards to stability. Using a detective approach a notification is sent to an administrator or maintainer when an update is detected. The recipient of the notification can review release notes and make a conscious decision whether the update should be implemented or not.

3.4.6 Cloud service

For this project Amazon Web Services has been the cloud service of choice. The main reason for this is that the product owner uses AWS, however AWS would most likely have been the service of choice regardless. At the time of writing this report AWS holds a significant majority of the market for cloud computing services and has a very well developed platform that offers cutting edge technological solutions. This project utilizes AWS Elastic Container Repository to host docker images, the containers spun up from these images are hosted by Fargate which in turn is managed by Elastic Container Service. For this project, despite the latency implications, it may be a solution to configure Fargate to close containers upon completion and only launch new instances on request, in order to reduce runtime costs as some latency is not an issue.

In the AWS ecosystem Fargate was chosen over EC2 and Lambda. EC2 would have required more maintenance and management. Lambda is better suited for functions and very simple applications that run for a short amount of time, which is not suitable for this project due to the need for relatively big packages that work together and run for significant amounts of time.

3.5 Work methodology

At the very start of the project it was decided to use SCRUM as the work method, mainly because both the team members were very familiar with the method. Additionally SCRUM is great to use in system development and makes it easy to make changes based on feedback from frequent meetings with product owner.

A couple weeks into the project the team decided to change from SCRUM to Kanban. This change was made because research and feedback showed Kanban would work better at a team with only 2 members. The main difference between the two methods of working is the fluid prioritizing in Kanban versus the sprints in SCRUM. Kanban does not use sprints and rather have a continuous backlogs which gets updates in tasks and prioritization frequently. SCRUM however uses sprints usually lasting 2-3 weeks, where the backlog and goal for the sprint is decided at a sprint planning.

With Kanban as the methodology the team set goals to work consistently with weekly meeting with the product owner. Run-able code was a priority, and the team would work to get early demos working to show product owner the project was heading in the desired direction.

The backlog was implemented with Atlassian's Jira which is the program the product owner mainly uses for backlog tracking. Alongside the backlog in Jira there was also created a roadmap which gave a visual of the project goals and wished deadlines.

For communications with product owner emails were used for all documentation. Documentation mainly includes notices and minutes of meetings, alongside relevant documents as vision document and requirement specification. There was also created a Slack channel for faster day to day communications.

3.6 Responsibility distribution

Due to the small size of the team most of the responsibilities have been shared. There are however a couple tasks that have been handled mainly by the same person throughout the project.

Olsen has dealt with most of the task management on our Jira roadmap and board.
Holt-Seeland has been dealing with the majority of the communication towards Telia and the supervisor from NTNU.

With regards to development, Olsen has dealt with JWT and CSRF authentication with the respective scripts and contexts needed to execute these authentication methods. Holt-Seeland set up the working environment with Kali, OWASP Zed Attack Proxy daemon and the python ZAP package. Development of the passive and active scanning was worked out as a team. Near the end Holt-Seeland shifted focus towards creating a depolyable docker image and implementation solutions while Olsen cleaned up the code.

4 Results

4.1 Scientific results

The thesis statement that was chosen for this project can be divided into the two different types of vulnerabilities. Security tests for the technical vulnerabilities are easier to automate with general tests that works on several applications, while the logical vulnerabilities is harder to automate on multiple applications, since every application works differently. Because of this, technical vulnerabilities have been the main focus of the automated tests made in this project, while logical vulnerabilities have been covered theoretically, and facilitated for further development.

When trying to reduce the need for manual tests it is important that the automated tests are reliable, while not notifying unnecessarily low security alerts. This could be regarded as spam, and reduce the chance of developers looking at the actual high severity alerts. The ZAP tool comes with it's own severity levels of security alerts, which have been used in this project for evaluating the severity. All the results of all severity levels are also included in a report which documents where the security vulnerability was found, what triggered it, and possible solutions for fixing the vulnerability.

Name	Risk Level	Number of Instances
SQL Injection	High	3
Application Error Disclosure	Medium	2
CSP: Wildcard Directive	Medium	2
X-Frame-Options Header Not Set	Medium	40
Absence of Anti-CSRF Tokens	Low	27
Application Error Disclosure	Low	1
Cookie Without SameSite Attribute	Low	1
Cross-Domain JavaScript Source File Inclusion	Low	2
Information Disclosure - Debug Error Messages	Low	2
X-Content-Type-Options Header Missing	Low	50
Information Disclosure - Sensitive Information in URL	Informational	14
Timestamp Disclosure - Unix	Informational	280

Figure 4.1: Example of alerts

To prove the reliability of the tests the project ran the tests against three different well known applications for security penetration testing, and the test have found high security alerts against all three. DVWA which is one of the applications used in this project also gives the option to change the difficulty of finding security vulnerability. The security scans created in this project manages to find high security alerts on all difficulties except the highest, which is named 'impossible'. It's also important for the security alerts to have reliable severity level, and security alerts with high severity is actually quite crucial to fix. The only high severity alerts found during these security tests have been Cross-Site scripting and SQL injection which have both been present on the OWASP Top Ten list for a long time.

How the security tests are ran against the application is also an important part of the process. It is important for the security tests to uncover possible vulnerabilities as fast as possible, and running tests CI/CD tests when new code is pushed will uncover new vulnerabilities right away. While this covers all possible vulnerabilities that can be find in a test environment, running security tests on an interval against applications in production will uncover possible vulnerabilities that may appear from using the application.

4.2 Engineering results

4.2.1 Vision document

The features mentioned in the vision document required security tests covering OWASP Top Ten, and for these tests to run when pushing new code. These requirements changed throughout the project where the team in agreement with the project owner concluded that not every point on OWASP Top Ten is possible to automate as a general solution, and the requirements changed to more specific technical vulnerabilities like SQL injection and Cross-Site scripting.

Further features included the tests were meant to be a general solution that could work with several teams with different applications. The feature of a general solution has been in focus throughout the whole project, and a lot of time has been invested into allowing several authentication methods to be used. This is a large part of the project, and creates a good foundation for further testing.

Scalability is also an important part of the project when several teams are involved. Using AWS Fargate as a cloud service will make the tests easily scalable, and should handle as many applications as necessary by setting up new containers for each team running tests. These containers will be removed after the tests are done, and results are saved.

As part of the non-functional features the tests are meant to be user friendly, so any developer can integrate the tests into their application, regardless of how experienced they are with security. To accomplish this the tests were created to rely on information provided in a single configuration file. This configuration file is required to give the tests necessary information about the application, and to help the developer fill out this information several templates are created. The config templates are created for three different applications, all using different authentication methods. Additionally all the variables in the file is also well documented with comments to explain the need and use for the variable.

The last requirement was unit testing for the project. Unit testing is provided to the scripts which are creating context and specifies how the scans are meant to run. Unit tests for the actual scans requires a running application to scan on, which after a discussion with project owner was not desired.

As this was a large project on a field in constant development, facilitating for changes was an important part of the project. This was kept in focus by creating a good foundation for security testing, making them a general solution. How the scripts were created was also affected by this focus, and the goal was having the main script detached from every security scan. The main script only focuses on retrieving the variables from the configuration file, and then gives these variables to other scripts running the scans. Future security scans to be developed can be created in the existing runScan.py script, or created in a completely new script. They only need to be imported to the main script, and called from there. This script setup will also make it possible for individual teams to create automated tests for their specific applications and then connect them to the main script.

4.2.2 Requirement specification

This project is mostly backend and does not have many frontend features beside the generated security report. This results in fewer user stories in the requirement specification, as there is little user interaction. The requirement specification was also created at a later stage in the project, since a lot of the time in the beginning was spent researching the relevant topics, and the requirements were set after the foundation was created. There was also made some changes at the end of the project after new decisions were made in prioritizing.

SQL Injection and Cross-Site Scripting vulnerability scans were prioritized first as these are some of the most common technical vulnerabilities, and have a high ranking on the OWASP Top Ten list. This requirement was met in the project, and with the scans covering these vulnerabilities, a lot of other injection vulnerabilities were also covered. These are not specified

in the requirement specification but included, among others, the vulnerabilities buffer overflow and CRLF injection. The full list is included in the system documentation A.3specification

JWT, CSRF tokens and a simple login as form-based login were the requirements for authentication methods. All of these methods are included, and can easily be specified in the configuration file.

After the scan is complete the results are saved in a HTML or XML report. This report contains detailed information about the different vulnerabilities found and suggestions on how to fix them. It is also prepared for notifications on Slack whenever a high security alert has been found by the scan, but it is not yet integrated into Slack. For now the notification only happens on the terminal, which is not too useful during automated scans, but it is only the actual Slack integration that is missing for notifications to Slack.

Implementation into AWS is one of the requirements specified. Due to a delay in access to AWS resources, this requirement has not been met. AWS access was provisioned at an early stage by Telia. However, what appears to be a conflict between the SSO tied to the NTNU provided e-mail accounts used by the team and Telia's SSO, getting access proved to be difficult. On delivery the project is at the stage where a docker image exists ready to be pushed to ECR. Actual implementation has not been completed.

With the last change to the requirement specification some security scans were replaced by authentication in order to build a foundation for further development. The removed security scans were brute force against admin user, and insufficient logging and monitoring. Insufficient logging and monitoring is present on OWASP Top Ten, and is recommended to be included in the tests at a later stage. The logging scan were removed after a discussion with product owner where adding more authentications was deemed more important than additional scans. Brute force attack against admin user was added to make sure the admins change their given default passwords, but was also removed after discovering ZAP does not have support for fuzzing in their API. This makes automated brute force attacks more difficult.

4.3 Administrative results

All hours spent in this project is documented in hour spreadsheets in the project handbook. Here we see that the hours spent is not as continuous as we would like. This is due to an imbalanced prioritizing between this project and another subject at the start of the project. The hour spreadsheets are well detailed with when and what the team spent time on.

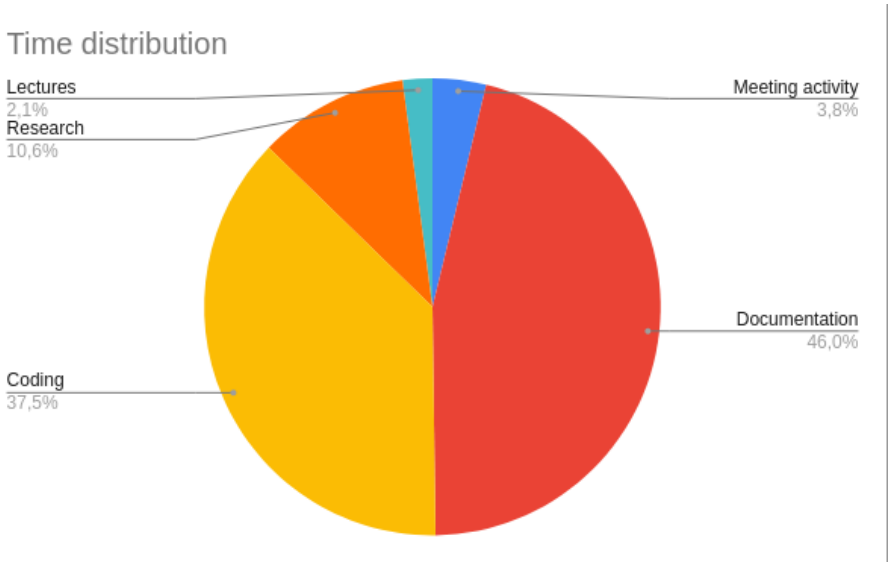


Figure 4.2: Time distribution

Here we see that most of the time was spent coding and documenting. Most of the time spent on research was spent at the start of the project, but there are also a lot of hidden research hours done while coding. Time has been spent on documentation throughout the whole project with frequent notice and minutes from meetings, but the report is where most of the documentation time has been spent. The weekly meetings does not take up too much time, but here again there are some hidden hours spent preparing for meetings, besides the meeting documentation.

The team was using Kanban and had a focus on working agile throughout the whole project. Kanban worked great with good use of a backlog board, and road map made in Atlassian's Jira. The backlog progress is being shown in the weekly logs the team has written throughout the project, located in the project handbook, and the road map taken from early April is shown below 4.3.

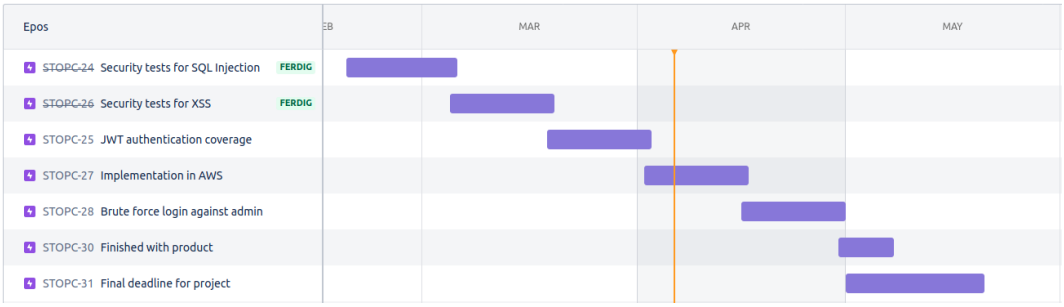


Figure 4.3: Road map from early April

This period is chosen since it shows the project progress during the project, and not at the end. As shown the JWT authentication is a little off the deadline, but everything was completed in time for product deadline. Brute force login against admin is also present at the road map, which was changed into form-based authentication when changes were made to requirement specification.

Weekly meetings were held with product owner where project progress was discussed. There was a goal to show frequent and early demos in these meetings to show project progress, however as the project progressed it became clear this project did not require particularly frequent demos, since a lot of the work done could not be shown as easily. Demos with running

scans and results were shown at an early stage, but additional changes to the foundation and configuration of the scans did not result in a different demo.

Another goal was for the team to be sitting together while working to make communications easier. This was done throughout most of the project, despite it being hard to get room for working on campus. The team ended up mostly working out of the apartment of a team member.

5 Discussion

5.1 Technical Results

At the start of the project a lot of research into new subjects and tools the team was not familiar with was needed. This resulted in less time being available to develop the product, which made it necessary to adjust the prioritizing and goals along the way. Since this is a large field that constantly progresses, creating a foundation for future development has been an important part of the project. Due to this, a broad foundation of various authentication methods has been a priority. Even though this work does not necessarily affect the scans themselves it enables a broader selection of applications to be scanned.

The project has proved it possible to automate security tests on several applications and facilitated good possibilities for further development. It is possible to automate security tests by setting up automated attacks that tries to breach common security vulnerabilities. If these attacks succeed it proves there exists a vulnerability. These tests will reduce the need for manual testing against several technical vulnerabilities, specifically injection type vulnerabilities, but there are still vulnerabilities that need test coverage, or manual testing.

5.2 Limitations

5.2.1 AWS

During the initial period of the project some effort was put into researching AWS, due to expectations that AWS access would be granted relatively fast. However, as a result of authentication issues using university e-mail addresses this took significantly longer than expected, with access to AWS being sorted very close to the project deadline. At this stage most of the formerly conducted research can be considered wasted as much of the research had to be repeated. In addition, the choice of AWS technologies changed during the course of the project.

5.2.2 OWASP Top Ten

OWASP Top Ten gives a good overview over different common vulnerabilities, and to cover these vulnerabilities was the project's first focus. However after the implementation of SQL Injection and XSS the focus was reconsidered, on the grounds that not all of OWASP Top Ten is easy to automate as a general solution. Several of the points on the OWASP Top Ten are considered logical vulnerabilities, and while some of them are automatable, it is hard to create a general solution that covers all points. This is because different applications often works very different logically, and to automate logical vulnerability scans requires in-depth knowledge about the specific application.

Some of the logical vulnerabilities mentioned on OWASP Top Ten are still very important, and quite common, so the tests are created with the possibility of further development. This makes it possible for each individual team to create their own scripts for testing their logical vulnerabilities, and attach these scripts to the already automated scans.

5.2.3 ZAP

As an open source penetration tool ZAP has provided good ways to automate the security tests. There are however some limitations to the tool, which have changed some of the priorities and goals of the project. The next priority after SQL Injection, XSS and building an authentication foundation was to create tests to brute force admin users. This is to make sure the users change their default passwords and the accounts are secure. After some research around brute force in ZAP it was discovered that fuzzing is not supported in the ZAP API. Using fuzzers for testing common passwords was the idea for the brute force, and while there probably are other options, the team and product owner decided to change the prioritization to create an even stronger foundation for further development instead.

5.3 Technologies

While using ZAP have given us good ways to automate scans, the opposition Burp Suite would have been able to provide the same features, and more in exchange for a premium fee. On certain areas ZAP may have limited the progress more than Burp Suite would have, but the fact that ZAP is open sourced while Burp Suite is not is enough to deem ZAP the correct choice. For product owner this was an important part of the choice, since the test then wont require a subscription to be used.

Python has also proven to be a good choice of programming language because of it's scripting capabilities, and existing version of python ZAP API. This helped greatly in making the code simple and easy to read, which will make it easier to further develop.

5.4 Society, environment and economics

System security has large impact on society in regards of how much we can rely on systems to do tasks. For the society to rely on systems it is important that the systems are secure, and know they have no been corrupted by any attacks made. The automated security tests made in this project will help improve overall security in the application they are testing, and they will also save time for developers to focus on other tasks. Which then can improve overall system quality and development progress.

While the tests cost money to run on AWS, they will still save money overall compared to developers running these tests manually. The alternative being not running tests at all to save money will again result in large risk of the systems being vulnerable to malicious attacks, which in turn may affect the company's reputation and their relations with customer. Harm to reputation may have long term implications that may be difficult to recover from. Having to weigh risk against cost and time is an issue under constant consideration throughout the development process. Using automated security tests will save time while reducing risk for security holes which saves money in the long run.

5.5 Future development

5.5.1 Authentication

Even though a lot of work have already been put into making the tests run with several different methods of authentication, there are more methods that can be relevant to include. Single Sign-On is one of the more modern authentication methods that is not already included in the project, and if the focus remains on a general solution adding SSO would be a good addition. Other options will depend on what the applications running these tests are using. Using the script based method in ZAP it is possible to replicate most of the authentication methods, even if they are not specifically mentioned in ZAP. To do this would require some work with the configuration file, adjusting it to fit the need of the new authentication method. However the CSRF token authentication method is already relying on script based authentication, so it should be possible to replicate most of the needed variables from that template.

5.5.2 Brute force

At a point brute force was on the list of tests that should be conducted, not in a full scale brute force manner but as a test that would check if default admin or weak admin passwords were in place. In the end it was bumped down the priority ladder in favour of completing several authentication methods. Part of the reason priority shifted away from brute force was that it is not implemented in the ZAP API, thus making it too time consuming to be worth prioritizing over authentication.

5.5.3 Insufficient logging and monitoring

Insufficient logging and monitoring is a point present on the OWASP Top Ten, and is possible test automatically. This could be done by running the scans already created, and then look for traces of the attacks in the logs. If there is not any traces of the attacks made, the logging is insufficient, and needs to be improved. For future work with general security scans that works with several applications this should be high on the list.

5.5.4 Logical vulnerabilities

Creating automated scans for logical vulnerabilities general enough for several application is hard to do. This is because applications work very different logically, and the testing the logic in two different applications is hard to do with the same tests.

Broken access control is a very common logical vulnerability which is possible to automate to some extent. There are security scans that know how to look if the access control is present, but the scan can not verify if the access control works as intended. Scanning to make sure the access control is present within all application is a good general solution, but will not remove the need for manual testing of the access control.

Since logical vulnerabilities varies greatly from application to application is it recommended that a developer with good knowledge about the application creates the automated tests, and these would probably only work with the specific application.

5.5.5 Frontend solution

This project have been focusing exclusively on the backend, and have possibilities in further development on a frontend solution. This frontend solution can for example provide a well organized way of showing test reports, and possibly compare reports to show progress.

The test should optimally run automatically on push of new code, or continuously on intervals, but if there is need for manually forcing a security tests this could also be an option in a frontend solution.

5.6 Administrative results

5.6.1 Process

The process worked well with Kanban and agile principles. A continuous Kanban board was used and kept updated throughout the whole project, and helped greatly keeping tasks sorted after priority. The team also had weekly meetings with product owner where task priority and progress of the project was discussed.

Another goal of Kanban and agile process is to focus on run-able code and demos. While this was in focus during development, it took longer time to get the first demo up than anticipated. This was mainly because the team was not familiar with several aspects of the project, and a lot of time was spent on research. Additionally, a lot of work was conducted on authentication configuration, which itself does not make for a good demo, but rather how the file is constructed and what is included in the configuration. Even though the first demo of the product took longer time than anticipated, there were weekly discussions about the research and progress of the project. According to feedback from product owner, these discussions all but qualified as a demo, giving them a clear picture of our progress and heading.

Alongside the project was another subject which ended in March. The balance between these subjects were at some points off, and at times too much time was spent on the other subject instead of this project. This resulted in larger amount of hours having to be spent later in the project, leading to a less consistent workflow. This inconsistency could result in unnecessary time being spent figuring out where the team left of last time, but was well mitigated with good use of the Kanban boards and the weekly discussions with product owner.

To optimize teamwork it was a goal to work from the same physical location as much as possible. Even though there were several restrictions on campus due to COVID-19, the team manages to follow this goal through almost the whole project. All meetings were held digitally since product owner was not located in Trondheim, but the team members spent most days in a members' apartment. This helped greatly, especially during the research phase where topics could be discussed for increased understanding.

5.6.2 Teamwork

The project has been largely unaffected by the ongoing COVID-19 pandemic. However, there have been incidents where one of the team members have gone into self-applied quarantine awaiting test results when experiencing potential symptoms. This has been of little consequence as we have been able to communicate through digital channels.
Outside of quarantine we have been working from the apartment of a team member. Due to this the team has not suffered a significant consequence from the restrictions to use of campus during the project.

The team members have been working together for a considerable amount of time over several projects and as of such have a well developed dynamic. Due to the team members having different areas of interest our combined skills and interests makes for a natural distribution of tasks and responsibilities. Despite of some gaps in productivity through the project communication has been good and there have been no conflicts.

Working on this project has been a valuable learning experience for both team members as we have been working more in-depth with several technologies and environments that the team previously had no or limited experience with. When faced with entirely new challenges the team has been able to work through by researching the topics both individually and as a team.

5.7 Positive elements

5.7.1 Product Owner

The product owner has provided good feedback throughout the project. Formal communication has been conducted through email exchanges while Slack has been used for less formal communication. Product owner has been reachable through both channels and communication has been easy. The product owner scheduled time on a weekly basis to conduct digital meetings. These meetings have provided an excellent opportunity to receive frequent live feedback on progress and work, which has proven to be very valuable.

5.7.2 Supervisor

The NTNU Supervisor of this project has also provided valuable feedback throughout the project. The supervisor has been easy to reach both through formal and casual channels, for questions of varying magnitude and arrangement of meetings. The meetings have been efficient and on point.

5.8 Project conclusion

The idea of this system was to create automated security tests that could be hosted on a public cloud in order to relieve some of the need for manual security testing. This goal was achieved as a general solution where the test can run several different applications by only specifying configurations to suit the application in question. This will reduce the need for manual security testing, but there are still several other vulnerabilities that are not covered by these general security tests, thus there will still be a need for manual testing against certain vulnerabilities such as logical vulnerabilities.

A docker image that contains all necessary packages and scripts has been made and can be pushed to ECR, from which it can be spun up by Fargate and managed by tasks in ECS. This

allows for excellent scalability and implementation with Telia systems. Config templates for three different authentication methods are ready to be stored on Amazon S3 from which the various teams can fetch copies, edit these copies and push to their respective containers.

References

[1] Atlassian (2021): *What are the differences between CI, CD, and CD?* https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment.

[2] Auth0 (2021): *Introduction to JSON Web Tokens* https://jwt.io/introduction.

[3] Auth0 (2021): *JSON Web Tokens* https://auth0.com/docs/tokens/json-web-tokens.

[4] Auth0 (2021): *Single Sign-On* https://auth0.com/docs/sso.

[5] AWS (2021): *AWS* https://aws.amazon.com/what-is-aws/?nc2=h_ql_le_int.

[6] AWS (2021): *AWS Fargate* https://aws.amazon.com/fargate/?nc=sn&loc=1.

[7] AWS (2021): *AWS Lambda* https://aws.amazon.com/lambda/.

[8] AWS (2021): *What is cloud computing* https://aws.amazon.com/what-is-cloud-computing/.

[9] Dhulam, Jit (2019): *JWT(Json Web Token) In Django REST API* https://medium.com/@ajitdhulam/jwt-json-web-token-in-django-rest-api-3056df2a4dfa.

[10] Imperva (2021): *Cross site request forgery (CSRF) attack* https://www.imperva.com/learn/application-security/csrf-cross-site-request-forgery/.

[11] Imperva (2021): *Cross-Site Scripting attacks* https://www.imperva.com/learn/application-security/cross-site-scripting-xss-attacks/.

[12] Klein, Amit (2005): *DOM Based Cross Site Scripting or XSS of the Third Kind* http://www.webappsec.org/projects/articles/071105.shtml.

[13] Netsparker (2021): *Understanding the Differences Between Technical and Logical Web Application Vulnerabilities* https://www.netsparker.com/blog/web-security/logical-vs-technical-web-application-vulnerabil

[14] Olan, Michael (2003): *Unit testing: test early, test often* https://www.researchgate.net/profile/Michael-Olan/publication/255673967_Unit_testing_Test_early_test_often/links/5581783608aea3d7096ff00c/Unit-testing-Test-early-test-often.pdf.

[15] OWASP (2021): *Active Scan* https://www.zaproxy.org/docs/desktop/start/features/ascan/.

[16] OWASP (2021): *Authentication Methods* https://www.zaproxy.org/docs/desktop/start/features/authmethods/.

[17] OWASP (2021): *Broken Access Control* https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control.

[18] OWASP (2021): *Contexts* https://www.zaproxy.org/docs/desktop/start/features/contexts/.

[19] OWASP (2021): *Cross Site Request Forgery (CSRF)* https://owasp.org/www-community/attacks/csrf.

[20] OWASP (2021): *Getting Started*
https://www.zaproxy.org/getting-started/.

[21] OWASP (2021): *Insufficient Logging Monitoring*
https://owasp.org/www-project-top-ten/2017/A10_2017-Insufficient_Logging%2526Monitoring.

[22] OWASP (2021): *Options AJAX Spider screen*
https://www.zaproxy.org/docs/desktop/addons/ajax-spider/options/.

[23] OWASP (2021): *OWASP Top Ten*
https://owasp.org/www-project-top-ten/.

[24] OWASP (2021): *Spider*
https://www.zaproxy.org/docs/desktop/start/features/spider/.

[25] OWASP (2021): *SQL Injection*
https://owasp.org/www-community/attacks/SQL_Injection.

[26] OWASP (2021): *Types of XSS*
https://owasp.org/www-community/Types_of_Cross-Site_Scripting.

[27] Portswigger (2021): *Cross-site request forgery (CSRF)*
https://portswigger.net/web-security/csrf.

[28] Portswigger (2021): *CSRF tokens*
https://portswigger.net/web-security/csrf/tokens.

[29] S, Wick (2019): *Single Sign On (SSO)*
https://medium.com/@wickyorama/single-sign-on-sso-69aad061e269.

[30] Swagger (2021): *Basic Authentication*
https://swagger.io/docs/specification/authentication/basic-authentication/.

[31] Zaproxy (2021): *Getting authenticated*
https://www.zaproxy.org/docs/api/#getting-authenticated.

A Documentation

A.1 Visions document

Project number: 063

Automation of information systems security on a Public Cloud

Vision Document

Version <1.0>

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
30/01/2021	1.0	First draft	Per Holt-Seeland, Mathias Olsen

Table of contents

1. Introduction

The purpose of this document is to define the scope of the project for automation of information system security.

The purpose behind the project is to create a continuous security system, which will make development and maintenance of relevant security easier. The product will contribute towards increased system security while relieving developers of the need to perform certain security tests manually.

1.1 References

https://owasp.org/www-project-top-ten/

2. Positioning

2.1 Problem statement

The problem of	Security testing being performed manually
affects	Developers
the impact of which is	Developers need to take the time to manually perform security tests. Potential low frequency of security scans as changes are implemented
a successful solution would	Allow developers to focus more on developing products while improving security

2.2 Product position statement

For	Telia Norge AS
Who	Desires automated security testing
The (product name)	Automation of information systems security on a Public Cloud
That	will automate security testing
Unlike	today's manual solution
Our product	does not rely on manual execution of security tests, freeing up time for developers to focus on other tasks.

3. Stakeholder and User Descriptions

3.1 Stakeholder summary

Name	Description	Responsibilities
Telia Norge AS	Owns the product	Sets requirements for the product, and offers guidance
Developers	Develops the product	Developer

3.2 User summary

Name	Description	Responsibilities
Supervisor	Gives advice and council when needed	Follow up the project and be prepared for meetings to give advice

Telia	Product owner	Provide feedback in order to make sure the project meets their expectations
Developers	Developers of the new system	Developing the new system after requirements from owner
Developers from Telia	Using the existing solution, and will be using the new system	None

3.3 Key Stakeholder or User Needs

Need	Priority	Effect	Current solution	Suggested solution
The users have a need to run continuous security tests as they develop or maintain software	High	Will free up time for developers to focus on other things, and will increase the frequency of security tests.	Manual security tests done by the developers	Automated security tests which runs on every commit.

3.4 Alternatives and competition

There are several libraries and tools for automation of security tests. Since security testing is such a wide field will the task of automating all the security tests be too large a task for most. Automated security tests can focus on different areas within the field, after the need of the owner. This product will be focusing on OWASP top ten. There are many tools and libraries for penetration testing,

4. Product overview

This product is intended to automate a broad suite of security tests in a manner that will run necessary tests depending on what is needed by the software to be tested. The product will be a general solution so the automated tests can be used for several teams in different fields. The tests will be able to integrate into already existing code for maintenance, and be used in developing new code.

4.1 Assumptions and dependencies

It is assumed that AWS Lambda will be used for less complex tests while AWS Fargate will be used for more complex tests. The different tests are supposed to be independent, and users should be able to select which tests they need to run in order to suit the specific needs of their project.

5. Product features

The system will be running tests on AWS and the code which will be located somewhere else. The system will cover OWASP top 10 and test these possible vulnerabilities automatically on pushing new code. It should be a general solution which could be used on several different development teams. The system should also be scalable and should be able to handle a large amount of tests running simultaneously if needed.

A possible expansion for the system could be increasing the amount of tests to include more than OWASP top ten. Another expansion could be to integrate AI to analyze logs and look for inconsistencies, which could possibly uncover security issues.

6. Non-functional features and other Requirements

It should be a user friendly solution, and be able to integrate into any and all developments which have use for security tests.

The tests should be reliable so there won't be a need to run manual tests double checking the automated tests. The automated test should rather flag one vulnerability too many, than one too few.

It should be possible to run several instances of the same tests on different code pushes. This will make the system scalable to handle several teams at once.

The tests should be covered by unit tests to make sure all the security tests give the correct results.

A.2 Requirement specification

Project number: 063

Automation of information systems security on a Public Cloud

Software requirement specification

Version <1.1>

Revision history

Date	Version	Description	Author
05/04/21	1.0	Creation of document	Per Holt-Seeland, Mathias Olsen
12/04/21	1.1	Added domain model	Per Holt-Seeland
2/05/21	1.2	Explanation to models, and some changes to user stories	Mathias Olsen

Table of contents

1. Introduction

This document is made to specify system requirements for the project Automated security tests on a public cloud. The requirements will include which tests are supposed to run and which security risks that should be covered. As this project also includes integration into Amazon Web Services (AWS) this will also be included in the requirement specification.

2. User Stories

```
As a tester
I wish to test SQL Injection[1] security
To evaluate how robust my application is against SQL Injection
attacks
```

- If I am a tester, I can perform automated security tests which test my application for SQL injection security holes.
- I can test my application for SQL injection by filling out a. context user, b. login username, c. login password, d. context name, e. target URL, f. API key, g. included URLs, h. excluded URLs, i. logged in regex, j. logged out regex
- I can not test the parts of my application that requires authentication, for SQL injection if I do not have valid login credentials

```
As a tester
I wish to test Cross-Site Scripting (XSS)[2] security
To evaluate how robust my application is against XSS attacks
```

- If I am a tester, I can perform automated security tests which test my application for XSS security holes.
- I can test my application for XSS by filling out a. context user, b. login username, c. login password, d. context name, e. target URL, f. API key, g. included URLs, h. excluded URLs, i. logged in regex, j. logged out regex
- I can not test the parts of my application that requires authentication, for XSS if I do not have valid login credentials

```
As a tester
I wish to utilize JWT[3] and CSRF[4] token authentication methods
So that I am able to test different applications
```

- If I am a tester, I need to be able to utilize different login criteria, namely JWT and CSRF tokens in order to test parts of my application that require authentication.
- I can choose between the different authentication methods by editing the config file

```
As a tester
I wish to utilize a simple form of login like form-based
authentication
So that I am able to test applications which uses a simple login
```

- If I am a tester, I need to be able to utilize form-based authentication to scan applications with a simple login.
- I can choose between the different authentication methods by editing the config file

```
As a tester
I wish to see a report of the performed tests which gives me an
overview of security holes
To easily determine if the application security needs to be improved
```

- If I am a tester, I need to be able to see a report from performed tests. The report needs to give a good and structured overview over potential vulnerabilities in my application.
- I can see this report by checking the html or xml file created after a performed scan.

```
As an admin
I wish to have the security tests running on AWS[(5)]
To remove the need for maintaining the test container as well as
reducing running cost
```

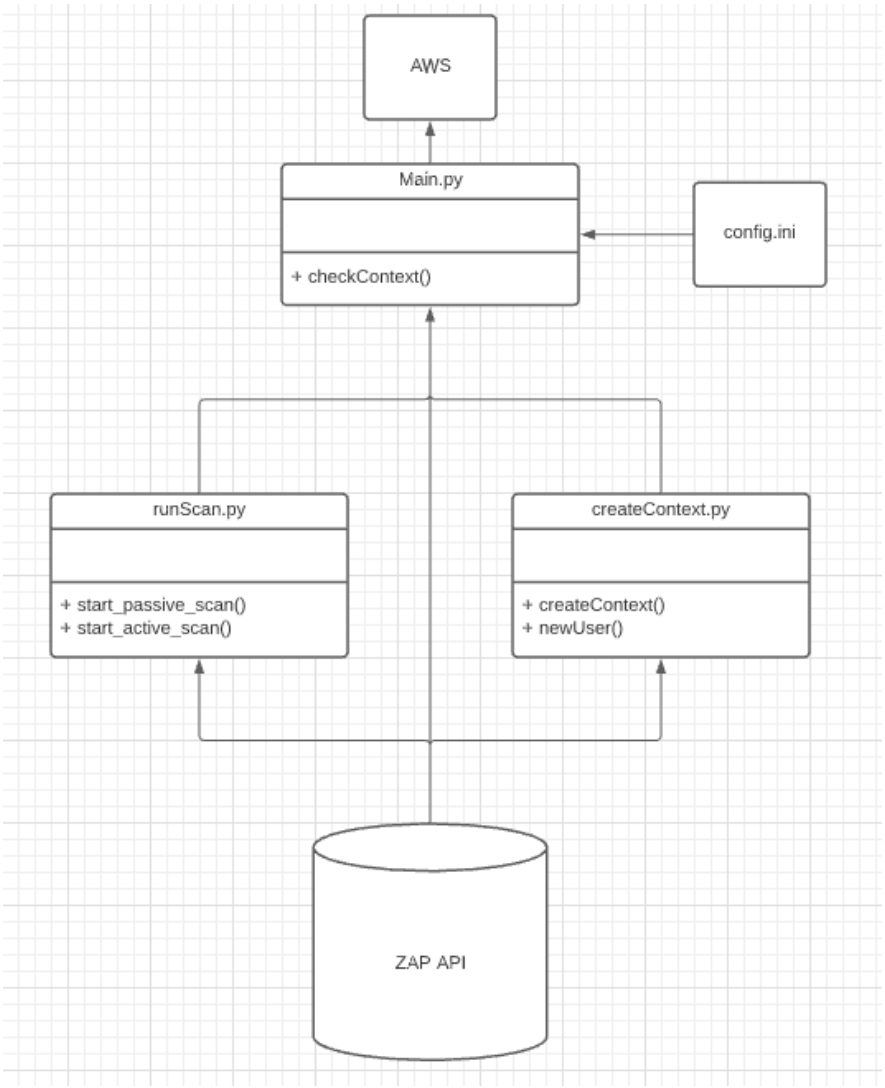
- If I am an admin, I need to be able to configure the security tests running on AWS.
- I can do this by logging into my Telia AWS account and find the cluster the scans are running on.

3. Domain model

Main.py is the main script which is run by the user. The script will receive all necessary variables from config.ini which contains application specific variables. After the config variables are received the main script will call methods from createContext.py which are checking for existing context within the ZAP API. This is to save time if this is the second security scan running from the same config file without a reboot. If the context is not found, another script from createContext.py will be called to create this context in the ZAP API.

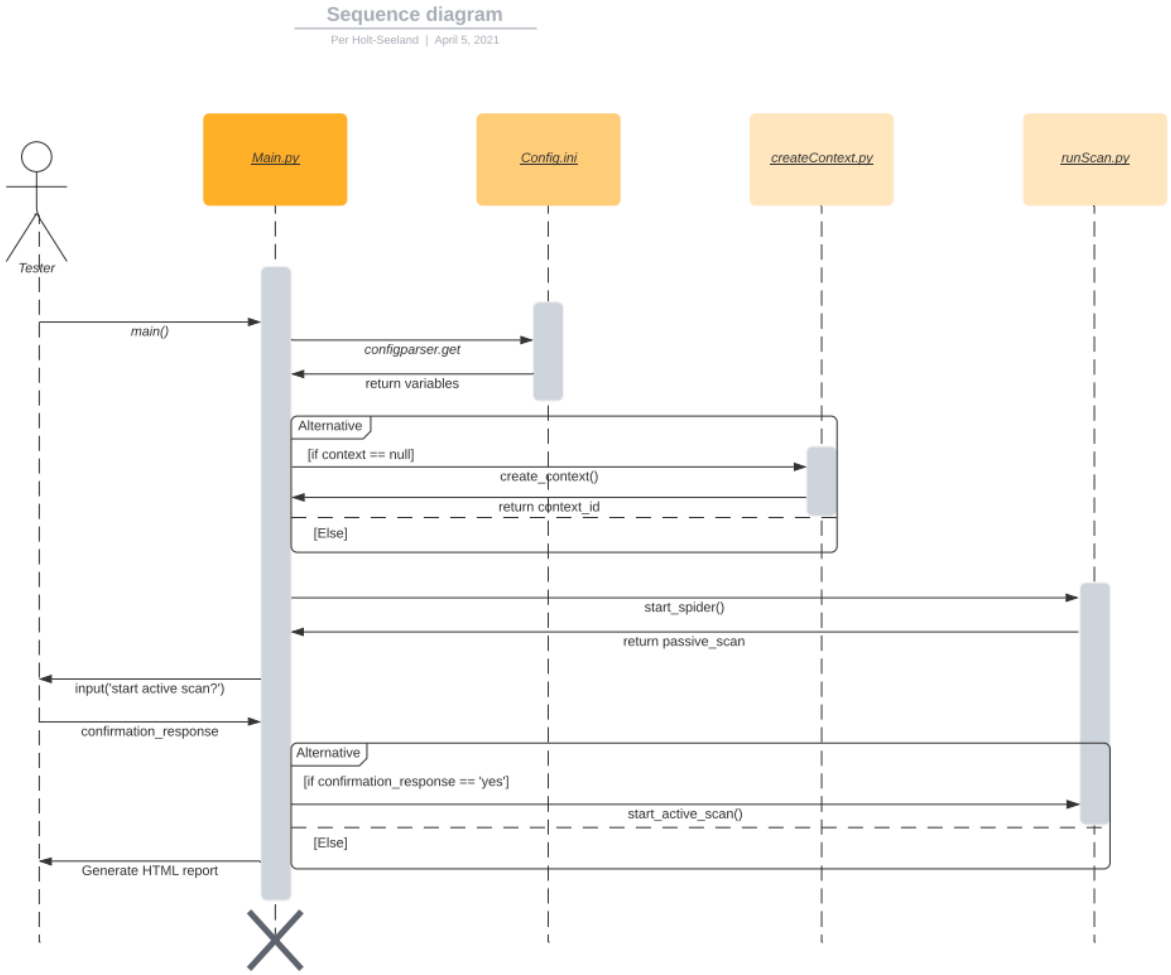
After the context and user is created from methods in createContext.py the main script will start running security scans. These scans are methods in runScan.py which are either passive scans or active scans. Passive scans are recommended to run first to check if the scans are attacking outside scope, if it is not the active scan can continue the scans.

The ZAP API is crucial for all the testing since it is the tool doing the actual scanning. All results from the scans will also first be saved in the ZAP API, and retrieved from there.



4. Sequence diagram

This sequence diagram shows how the main script runs from start to end. What the different script does is stated above, but this script also specifies how the user interacts with the script. The user only needs to start the main script, and then only required to say yes to the active scan prompt. This prompt is not needed for automated security scans since there is not an user to confirm the prompt, but it is recommended to perform a test with the prompt beforehand. After the scan is complete the script will save a html report with all the results.



5. References

(1) OWASP Foundation. (2021, March). *SQL Injection*. Retrieved from OWASP: https://owasp.org/www-project-top-ten/2017/A1_2017-Injection

(2) OWASP Foundation. (2021, March). *Cross-Site Scripting*. Retrieved from OWASP: https://owasp.org/www-project-top-ten/2017/A1_2017-Injection

(3) Auth0. (2021, March). *JWT*. Retrieved from JWT: https://jwt.io/introduction

(4) PortSwigger. (2021, March). *CSRF token*. Retrieved from PortSwigger: https://portswigger.net/web-security/csrf/tokens

(5) Amazon Web Services. (2021, March). *AWS Introduction*. Retrieved from AWS: https://aws.amazon.com/what-is-aws/?nc2=h_ql_le_int

A.3 System documentation

Automatic security tests on a public cloud

System documentation

Version 1.1

May 20, 2021

Revision history

Date	Version	Description	Author
10/05/21	0.1	Start of document	Mathias

Contents

1 Introduction

This document provides a general overview of the system developed by this project and its relevant technical documentation. It describes the structure of the project, how to install and run it as well as the proposed AWS implementation.

2 Project structure

2.1 automatic_security_tests/

Root directory for the automatic security tests. Contains the README.md file which explains how to run the tests, and a htmlReport.html report which contains the latest made security scan report. All other directories are located in this direction.

2.1.1 security_tests/

All security test scripts are located in this directory. There are three different scripts, which are main.py, runScan.py and createContext.py. There is also a config.ini file which are used to give necessary variables to main.py. In this directory there is also the unit tests created for createContext.py. These tests are written in pytest, and can be run by writing py.test when in the directory.

2.1.2 config_templates/

This directory have all the templates for the configuration file. These templates are made for the applications bodgeit, juice shop and dvwa. All these applications are using different authentication methods so gives a good indication on how to replicate these methods in other applications.

2.1.3 authentication_scripts/

In this directory there are the necessary scripts used in specific authentication methods. These scripts are necessary for using jwt tokens and csrf tokens.

3 Cloud service

Disclaimer: this section must be seen as theoretical. Due to the late stage at which AWS access was successfully granted, no implementation was completed. As of such, this section will describe the general idea of the planned implementation.

A Dockerfile is included in the product repository. The Dockerfile builds an image based on Kali Linux, installs the required packages and uploads the project code to /root/ in the docker image. This docker image can be pushed to ECR where it will be stored for Fargate to run it. Config file templates can be stored on S3 for teams to read and edit to suit their application, before submitting them to their respective container prior to running a scan. Script file templates may also be stored on S3 in case specific changes need to be made to suit an application, even though the scripts should be general enough to work with information provided in the config files.

4 Installation and running

Necessary dependencies for running the tests locally are running ZAP in daemon or UI, python3 and python zap API.

The python zap API can be downloaded with pip with *python-owasp-zap-v2.4 0.0.18* as parameter.

ZAP can be downloaded on their official website: https://www.zaproxy.org/download/

Additionally an application to run the tests on are needed. Templates are created for DVWA, Juice-Shop and Bodge-it. These can be downloaded and ran locally, or fetched as an image through docker.

4.1 Start of application

When all dependencies are downloaded the tests can be ran through main.py in /security_tests. In this directory there is also located an config.ini file which contains necessary variables to run the tests. This file contains on default variables for bodge-it, but templates for the other applcations can be found in /config_templates.

ZAP can start in daemon by running zap.sh located in /usr/share/zaproxy. The daemon can be ran by running *./zap.sh –daemon*. Additionally the API key needed from the ZAP tool can be changed by running *./zap.sh –daemon -config api.key="change-me"*. The API key needs to be defined in the configuration file, and can also be found through the UI by using key combination *ctrl + alt + O*, under the API tab.

By running main.py while ZAP is running in daemon or a proxy the security tests will start. First a passive scan will run, and after the passive is confirmed the user need to confirm prompt to continue running the active scan. After the active scan is complete the results will be saved as htmlReport.html in root directory.

4.2 Configuration

These are all configuration options present in config.ini.

`User`: Name of the user in context. Does not have to be the same as credentials

`Username`: Username or email credential necessary for login

`Password`: Password credential necessary for login

`ContextName`: The name of the context. Used for checking if the context already exists, if changes are made to config,ini while ZAP is running the name needs to be changed, or ZAP needs to restart.

`TargetURL`: Starting URL for the scans to test

`APIKey`: Necessary API key for using the ZAP tool. Can be found in ZAP UI.

`UseAjax`: Defines if the passive scan is performed by a regular spider or ajax spider. Ajax spider is recommended for applications using React.js, Angular.js or Vue.js.

`IncludeInContext`: Regex for which URLs are to be included in the scan.

`ExcludeFromContext`: Regex for which URLs are to be excluded from the scan.

`LoggedInRegex`: Regex to specify when the user is logged in or out. Needed for the scans to know when to perform login action. Response messages will be searched for match to regex.

`LoggedOutRegex`: Works the same way as `LoggedInRegex`. It is not necessary to use both `LoggedInRegex` and `LoggedOutRegex`, but both can be used at the same time.

`PostData`: Post data for login POST request. Usually contains username and password, but may also contain csrf token or other parameters.

`LoginUrl`: URL used for sending login request.

The next three configuration options are to define authentication methods. Set one to True and the other two to False. `UseBasicAuth`: Set to True to use BasicAuth authentication method

`UseCSRF`: Set to True to use csrf tokens authentication method

`UseJWT`: Set to True to use JWT authentication method.

`jwtScriptName`: Name of the JWT script for adding token to requests after login. Default is jwtScript.js and does not need to be changed unless the name of the script is changed.

`jwtScriptLocation`: Location of the JWT script. Default is location within docker image. Needs to be changed if used outside docker, depending on location of the script.

`csrfScriptName`: Name of csrf script to fetch csrf token from login URL. Default is csrf_login.js and does not need to be changed unless name of the script is changed.

`csrfScriptLocation`: Location of the csrf script. Default is location within docker image. Needs to be changed if used outside docker, depending on location of the script.

`csrfTokenField`: Name of csrf token field on site. Is not used unless authentication method is csrf.

`MaxSpiderDuration`: An upper limit for duration of passive scan. Is recommended to add when using Ajax spider. Set 0 for no restrictions.

`MaxSpiderDepth`: An upper limit to depth of passive scan. ZAP default is 10.

5 Documentation of source code

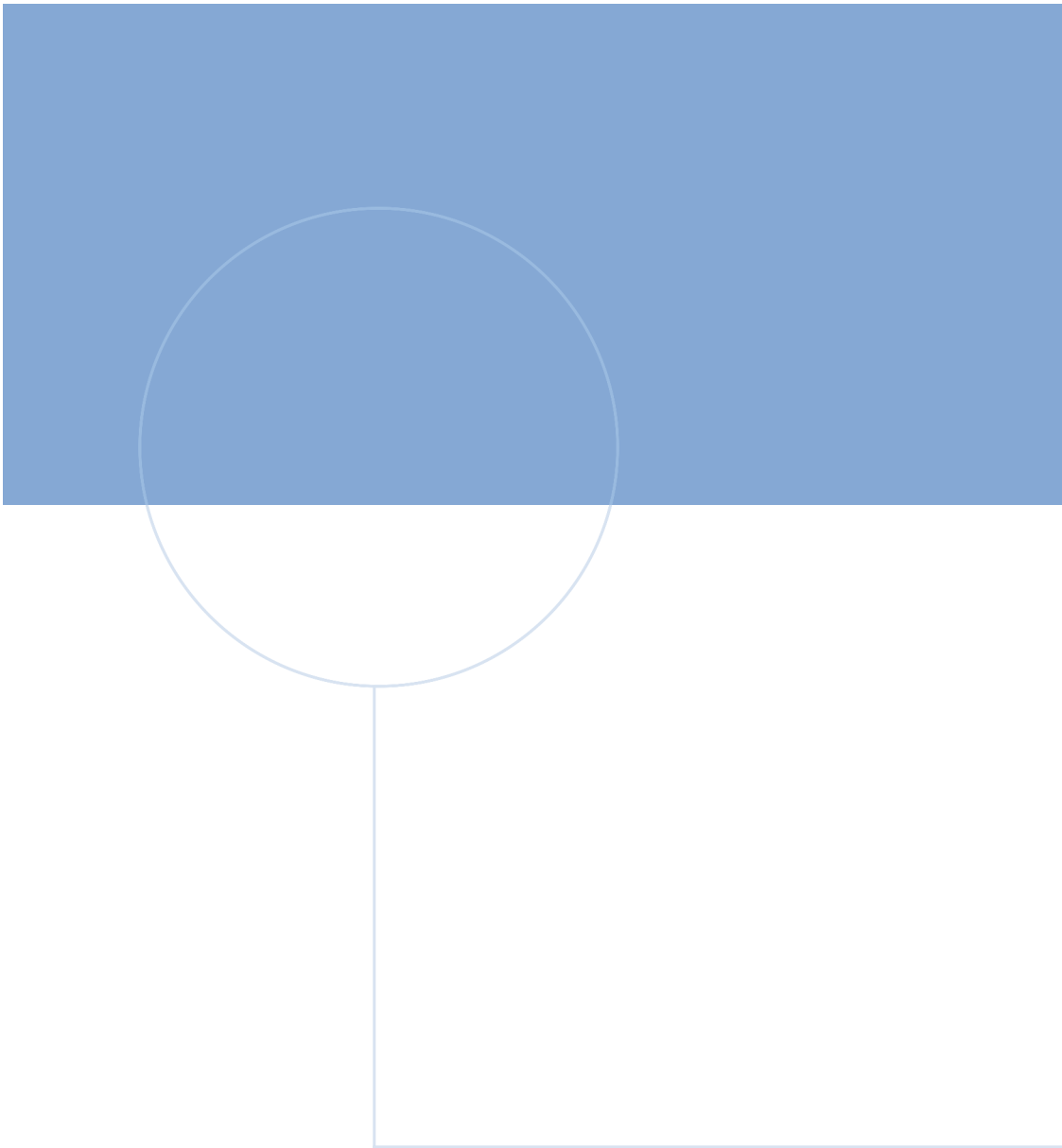
All code is documented with comments within the script.

6 Testing

6.1 Unit testing

Unit testing is performed with pytest. Only createContext.py is covered with unit tests since unit tests for runScan.py requires a running application that can be attacked. To run unit tests for createContext.py run py.test when in /security_tests directory.

References



NTNU

Kunnskap for en bedre verden